Disclosure BUR8-2000-0250

Prepared for and/or by an IBM Attorney - IBM Confidential

Created By: William Devine **Created On:**
Last Modified By: wpts1 wpts1 **Last Modified On:**

Required fields are marked with the asterisk (*) and must be filled in to complete the form .

***Title of disclosure (in English)**

On-Chip Logic Analyzer ("OCLA")

Summary

Status	Final Decision (File)
Docket Family	BUR9-2000-0199
Processing Location	BUR
Functional Area	CBB Central Processing Units ... 201
Attorney/Patent Professional	Richard Henkler/Burlington/IBM
IDT Team	Marc Faucher/Burlington/IBM
Submitted Date	
Owning Division	MD
Incentive Program	
Lab	Beers
Technology Code	201
PVT Score	No PVT score has been calculated.To calculate a PVT score, press the 'Calculate' button.

Inventors with Lotus Notes IDs

Inventors: Steven Parker/Fishkill/IBM, Joseph Marsh/Poughkeepsie/IBM, Frank Scanzano/Poughkeepsie/IBM, William Corti/Fishkill/IBM, Robert Kenny/Fishkill/IBM

Inventor Name	Inventor Serial	Div/Dept	Inventor Phone	Manager Name
> Parker, Steven C.	988745	29/NYEA	N/A	Schuh, Brian J.
Marsh, Joseph O.	460799	7R/WYIA	N/A	Miller, William R. (Bill)
Scanzano, Frank	982561	7R/WYIA	N/A	Miller, William R. (Bill)
Corti, William D.	512577	29/NYEA	N/A	Schuh, Brian J.
Kenny, Robert Jr.	831651	29/NYEA	N/A	Schuh, Brian J.

> denotes primary contact

Inventors without Lotus Notes IDs

IDT Selection

IDT Team:	Attorney/Patent Professional:
Marc Faucher/Burlington/IBM	Richard Henkler/Burlington/IBM

Response Due to IP&L :

***Main Idea**

To view the main idea for this disclosure, click on this doclink --->

***Critical Questions (Questions 1-9 must be answered)**

***Question 1**

On what date was the invention workable **ıse format the date as MM/DD/YYYY**
(Workable means i.e. when you know that your design will solve the problem)

***Question 2** — ○ Yes ● No

Is there any planned or actual publication or disclosure of your invention to anyone outside IBM?

If yes, Enter the name of each publication or patent and the date published below.
Publication/Patent:
Date Published or Issued:

Are you aware of any publications, products or patents that relate to this invention? — ● Yes ○ No

If yes, Enter the name of each publication or patent and the date published below.
Publication/Patent:TI Patent -see Steve & Whitepaper by TI -see teve
Date Published or Issued:

***Question 3** — ○ Yes ● No

Has the subject matter of the invention or a product incorporating the invention been sold, used internally in manufacturing, announced for sale, or included in a proposal?

Is a sale, use in manufacturing, product announcement, or proposal planned? — ○ Yes ● No

If Yes, identify the product if known and indicate the date or planned date of sale, announcements, or proposal and to whom the sale, announcement or proposal has been or will be made.
Product:
Date:
To Whom:
If more than one, use cut and paste and append as necessary in the field provided.

***Question 4**

Was the subject matter of your invention or a product incorporating your invention used in public, e.g., outside IBM or in the presence of non-IBMers?

If yes, give a date. **Please format the date as MM/DD/YYYY** — ○ Yes ● No

***Question 5** — ○ Yes ● No

Have you ever discussed your invention with others not employed at IBM?

If yes, identify individuals and date discussed. Fill in the text area with the following information, the names of the individuals, the employer, date discussed, under CDA, and CDA #.

*Question 6 Was the invention, in any way, started or developed under a government contract or project?	○ Yes ● No ○ Not sure
If Yes, enter the contract number	

*Question 7 Was the invention made in the course of any alliance, joint development or other contract activities? If **Yes**, enter the following (**In English**):	○ Yes ● No ○ Not Sure
Name of Alliance, Contractor or Joint Developer	
Contract ID number	
Relationship contact name	
Relationship contact E-mail	
Relationship contact phone	

*Question 8 Have you, or any of the other inventors, submitted this same invention disclosure or similar invention disclosure previously?	○ Yes ● No
If Yes, please provide disclosure number below:	

*Question 9 Are you, or any of the other inventors, aware of any related inventions disclosures submitted by anyone in IBM previously?	○ Yes ○ No
If Yes, please provide the docket or disclosure number or any other identifying information below:	

Question 10
What type of companies do you expect to compete with inventions of this type? *Check all that apply.*

- ☐ Manufacturers of enterprise servers
- ☐ Manufacturers of entry servers
- ☐ Manufacturers of workstations
- ☐ Manufacturers of PC's
- ☐ Non-computer manufacturers
- ☐ Developers of operating systems
- ☐ Developers of networking software
- ☐ Developers of application software
- ☐ Integrated solution providers
- ☐ Service providers
- ☒ Other (Please specify below)

semiconductor manufacturers

Question 11
If the invention relates to a product or service that is outside the scope of your business unit, please recommend IBM business unit(s), IBM location(s) or individual(s) within IBM that you think would provide

a good evaluation of your invention:

Patent Value Tool (Optional - this may be used by the inventor and attorney to assist with the evalu
(The Patent Value tool can be used by the inventor(s) to determine the potential licensing value of your invention.)
No PVT score has been calculated. To calculate a PVT score, press the 'Calculate' button.

Market

What is the anticipated annual market size (in dollars) that will be captured by your invention?

CLAIMS

Question 1 - How new is the technical field?

Question 2 - How central is the invention to the product(s) which might be expected to contain the invention?

Question 3 - What is the scope of the claim?

PORTFOLIO NEED

What are the portfolio needs in the area of your invention?

EXPLOITATION & ENFORCEMENT

Question 1 - How easily can the use of the invention by a competitor be detected?

Question 2 - How easily can the use of the invention be avoided by a competitor?

BUSINESS VALUE

Question 1 - What percentage of the companies producing products in the field of this invention might use this invention?

Question 2 - What is the value of this patent to current or anticipated Alliance Activity between IBM and other companies?

Question 3 - What is the value of this patent to current or anticipated Technology Transfer Activity between IBM and other companies?

Question 4 - Does it result in prestige to IBM?

Evaluation

This evaluation was entered by **Marc Faucher/Burlington/IBM**
Team Evaluation What is the team's evaluation of this disclosure? Search
Date rated :
Evaluation Comments

Final Evaluation History: Search	Who made the final evaluation: Marc Faucher/Burlington/IBM	Final evaluation date:

Search Information

Date sent:	Target completion date:	Search Results Received date:
Who was the search sent to (This area is to designate a Local Searcher name or WAIPL):		
Send search request to: Joy Doyle/Arlington/IBM@IBMUS Abdi Dirie/Arlington/IBM@IBMUS	Search Type: ☒ Patentability ☐ Clearance ☐ Validity ☐ State of Art	
Features to be searched: Abdi: Please provide us with 2 copies of the search results. Feature: Ability to test a microprocessor embedded on a chip with other components. A beffer is used to store the signals being sent to and from the microiprocessor during test mode. A software mask can also be used to stop the execution of the microprocessor upon the occurrence of certain conditions. The stopping can be instantaneous or delayed.		

Search Office Information

Target completion date:	☐ Search has been delayed	Ship/Return date:
Search Conducted By		
Comments		

Final Decision

This decision was entered by **Cynthia Hill/Burlington/IBM**

Decision: File	Status: N/A
PPM Area: 200 - Computer Architecture	Attorney Rating: 2
Date of Final Decision :	

Additional filing information

Planned Filing date:
Filing comments:

Dates have been entered in the format MM/DD/YYYY

Additional decision comments

Final Decison History

Entered on by Cynthia Hill
File N/A Docket Family: BUR920000199

Post Disclosure Text & Drawings

Enter any additional information relating to this disclosure below:

(Form Revised 12/17/97)

Main Idea for Disclosure BUR8-2000-0250

Prepared for and/or by an IBM Attorney - IBM Confidential

Archived On:

Title of disclosure (in English)

On-Chip Logic Analyzer ("OCLA")

Idea of disclosure

1. Describe your invention, stating the problem solved (if appropriate), and indicating the advantages of using the invention.

Bill Devine , MD IAD , E. Fishkill , assisted with this disclosure . Contact Bill at tl533-1532.
attn: IPLAW please add William Devine as an IDT member for this disclosure

INTRODUCTION

Inventor comment

The DSP-based servo controller presented several unique challenges as it was one of the first system on a chip ("SOC") applications we have approached. On this chip were several internal busses (DSP<->SRAMs) which had no external visibility for bring-up/debug. To complicate issues further, a DSP features a Harvard based architecture which means up to 3 memory busses could be active at any one time.

Attached is a document describing our unique solution to the problems associated with the bring-up/debug of Harvard architecture systems on a chip. We call this design the On-Chip Logic Analyzer ("OCLA"). It includes putting a logic system with the functionality of a logic analyzer on a target chip. In addition we offer modes to allow debug of real-time software elements.

We believe this is a unique approach to the problems facing all real-time SOC designers. Texas Instruments has also offered their own solution to these problems, however we feel our approach is superior to theirs.

This design effort was completed in Fishkill under manager Brian Schuh .The project was terminated and then later
picked up by IBM Yasu Japan.

PRIOR ART

IAD Comment (Bill Devine)

This invention has potential to be a very valuable patent for licensing and avoidance of the recently issued issued
patent BY TI attached here .....

METHOD FOR TESTING AN INTEGRATED CIRCUIT WITH USER DEFINABLE TRACE FUNCTION

Inventors: Gary L. Swoboda, Sugarland; Eric J. Stotzer, Houston, both of Tex.

Assignee: Texas Instruments Incorporated, Dallas, Tex.

PAT5884023.p

Patent Number: 5,884,023

Date of Patent:

[57] **ABSTRACT**

A method for testing a digital processor **11** in which a test port **1149** is used to transfer trace data from the digital processor to a test host processor **1101** under control of a user definable program which executes in response to predetermined events on the digital processor. Trace data is gathered while an application program loaded in program memory **61** is executed by the digital processor. Trace data is temporarily stored in a trace region **99** of data memory **25** by user definable code which is executed in a background manner by the digital processor in response to trigger events. The trigger events are also enabled by user definable code which enables various portions of analysis hardware **1217**. Trace data is transferred from the digital processor to the test host processor through test port **1149** by sending a notification signal to the test host processor by means of message passing register **1216**. The digital processor then monitors the message passing register for a handshake signal from the test host processor. When a handshake signal is received, trace data is written into the message passing register by user definable code in a background manner and transferred to the test host processor.

19 Claims, 14 Drawing Sheets

What is claimed is:

1. A method for testing a digital processor, said digita processor having a memory portion and a test port, com prising the steps of:

defining an area of said memory portion as a trace region

executing an application program on said digital proces sor until a preselected event occurs;

storing at least one data item in a location in said trac region in response to said preselected event by sai digital processor executing a monitor program include with said application program, such that said digita processor continues to execute said application pro gram after said data item is stored, wherein said moni tor program is operable to perform additional process ing and may be varied by loading a new applicatio program; and

transferring said data item to a test host processor throug said test port such that said digital processor continue to execute said application program after said data iten is transferred.

PROBLEM SOLVED BY YOUR INVENTION

What was the problem your invention solves?

1. Intended implementation environment

Real-time microprocessor and digital signal processor based systems pose special problems not present in traditional microprocessor based system. One of the major problems has been in the way these systems are tested and debugged. Traditional emulator schemes such as the JTAG system have relied on the ability to start and stop the processor through the use of a variety of mechanisms. These mechanisms have included setting software breakpoints and single stepping through the machine code. A major draw back of these techniques for real-time systems is the fact the processor must be stopped once a breakpoint is reached or the instruction is executed. In real-time systems it is impractical to stop the processor. For example, in a situation where the DSP is being used as a digital motor controller, stopping the processor could drive an invalid control signal to the actuator with a disastrous result. Therefore test and debug techniques where the processor is not stopped need to be developed.

2. Hardware debug issues

Hardware debug of real-time digital signal processor systems presents additional problems. The architecture of modern DSPs consists of multiple address and data busses for both program and data memory spaces. Numerous combinations of these busses could be active at any one time. In `system on a chip' situations where the memory is located on the same chip as the processor core, there is no practical way to view the activity of these busses as you would in a traditional system which has off-chip memory with a logic analyzer. For this reason it is almost impossible to debug a system like this using existing methods.

3. Software debug issues

Development of digital signal processing algorithms also requires special techniques. In many cases the algorithm developer would like to monitor isolated blocks of his or her DSP software design. Reasons for monitoring input-output behavior of these block may be for checking fixed-point performance of the implemented system versus preliminary simulation results, watching for internal overflow or saturation of the MAC within the system, etc. Previous attempts at monitoring these signals have required software hooks to trace these signals, and external processor bus bandwidth, which may or may not be available, to off load them to a PC or workstation. If the signal desired to be monitored changes, it would likely take a code recompile. Therefore it would be good to have a scheme

to monitor signals in a DSP without making code changes and without using external bus bandwidth to off load the data.


Disclosure99.lw

Bill,

I conducted another search through the IBM Patent server found the patent which I believe covers the approach TI is using for real-time emulation. The date of this patent is - so I do not believe it was available when I conducted my original search. This new patent is the only one which I have found which is remotely similar to solving problems like the OCLA.


PAT5884023.p

However, TI did publish an initial white paper) which described an implementation using the invention in the above patent. These ideas are marketed under the trade name RTDX (Real-Time Data eXchange). At the time I was initially presenting the OCLA to the IP law department, I did illustrate the notable differences of the OCLA vs. RTDX as well as the key novelties. I described these in a paper I wrote:


ocla v. rtdx.lwp

I feel the development of TI's RTDX and IBM's OCLA represent independent solutions to a similar problem. I hope that as you learn more about each of these systems you too will realise they are very different.

1.0 A Comparison of IBM's OCLA with TI's RTDX

OCLA Team
(Contact: Steve Parker, T/L 532-2838)
IBM Microelectronics Division
Dept. NYEA / 321-2
1580 Route 52
Hopewell Jct., NY 12533

1.1 Overview of RTDX

On Feb Texas Instruments accounced a new offering to compliment their DSP solutions called Real-Time Data Exchange ("RTDX") New Texas Instruments Development Technology Provides Window into Real-Time TMS320 DSP Systems. Semiconductors in the News. Febr < http://www.ti.com/sc/docs/news/1998/98004a.htm>.. The motivation for the RTDX is similar to that which inspired the OCLA, the need to evaluate DSP target applications in real time.

RTDX is based on a software API which is used by the DSP code to manually place trace points in the target code. These trace points can be used to do two things, first interal variables of the DSP can be monitored through the manual placement of trap routine calls in the target applicaiton. Second, new data can introduced into the system.

The following are major features of the RTDX system:

- Will work on existing TI architectures
- Serial communications through JTAG serial port
- 8 kilobytes/second throughput
- Based on an DSP RTDX software library
- Allows both "real-time" upload and download of samples into target systems (note: this is currently too slow for modern applications).

1.2 Operation of RTDX

Based on the white paperReal-Time Data Exchange. Texas Instruments White Paper: SPRY012. Digital Signal Processing Solutions. February 1998. < http://www-s.ti.com/sc/psheets/spry012/spry012.pdf > TI released on RTDX the operation of this system works through the following architecture:

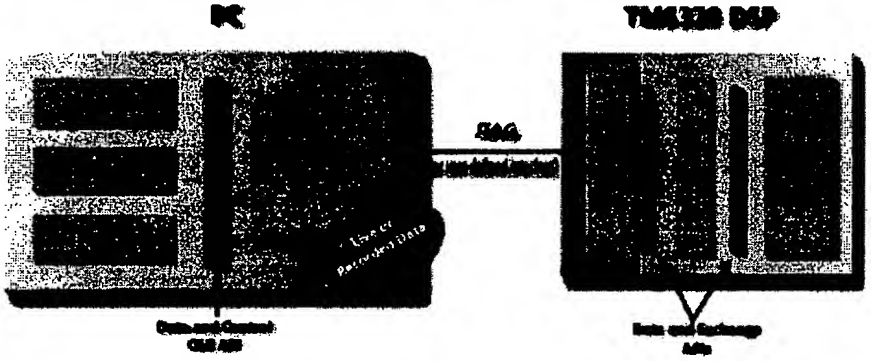


Figure 1: RTDX data flow (Taken from Texas Instruments White Paper: SPRY012)

From the target side, the DSP coder will make use of routines in the RTDX library like
RTDX_Data_Write( ) and **RTDX_Data_Read( ).** These two routines queue up an exchange to be processed over the JTAG emulation logic. In order to monitor signals real-time, calls to these routines must be made once per sample period. With the current speed limitation imposed by the JTAG system, TI has reported transfer rates of 8 kb/s. Since we are typically running with 16 bit samples this would impose a 4000 samples per second rate which we could monitor real-time systems.

DSP sampling rates (A/D, D/A conversions) for our Sailfin/Marlin operation are currently at about 10,000 samples / second. The present TI version of CRANE has the JTAG logic already around the DSP core, so it would be very possible in the near future that RTDX could even be used with our exisiting CRANE parts. As an example, lets explore this possibility.

1.2.1 Use of RTDX as a DSP Code Development Aid

As with most DSP applications, the micro-code which runs the Sailfin/Marlin drive is timed off a main-loop interrupt of 10 kHz rate. RTDX would be almost useless, since it could capture nothing even close to this rate. This system could however be used to monitor slow real-time events, one example might be an interal software based state machine.

Mentioned in the white paper for the RTDX was the possibility to use a

serial port other than the JTAG, (perhaps one which would even match the performance of the DSP). Since in order for a higher performace serial port to operate it would most likely still need to perform cycle stealing from the processor - this would have an impact on code timing as well as MIPS available to the running algorithms.

The throughput of a typical DSP system in a high performance would be something like:

2 * (10 kilosamples/second) * (2 bytes/sample) = 40 kilobytes / second

Assuming we are using 16 bit samples and would like to monitor the input-ouput behavior of the system.

If the RTDX was matched up with a system capable of handling the through-put of an application like a high performance disk-drive, the software trapping hooks could be installed around a piece of DSP code. To illustratate this we will looks at the piece of sample code from IBM disclosure (FI898-0017) On-Chip Logic Analyzer for Real-Time Systems. IBM internal disclosure (FI898-0017). Janurary 13, 1997..

Most digital signal processor software development methodologies will use a block diagram as a high level visualization of the system architecture. These block diagrams are used for high level software simulation of DSP components, as well as the blueprint for the assembly language coding of the signal flow. To illustrate this point lets propose that we would like to look at the input-output relationship of a finite impulse response filter deeply embedded within the system (Figure 4).

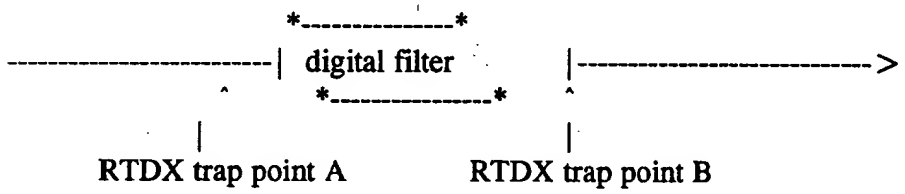


Figure 2: Software capture points around a DSP filter structure

To implement this digital filter in with RTDX tracking, it might look something like this (Finite Impulse Response filter in pseudo assembly):

PC	Label	Instruction
1000	PC_value_1:	LOAD input into a DSP register (possibly the accumulator)
1002		**RTDX_Data_Write(New_Input)**

```
1004                    LOAD starting points of addressing registers

1006                    LOAD repeat loop register with N
1008                    CLEAR multiply accumulate register
100A                    MULTIPLY-ACCUMULATE N times, update
pointers
100C                    SHIFT to normalize result (if necessary)
100E  PC_value_2:       STORE result
1010                        RTDX_Data_Write( New_Output )
```

*** BOLD indicates RTDX API function call.**

Figure 3: Pseudo assembly implementation of filter structure in Figure 2.

When the A/D conversion is made and the DSP receives the interrupt we will enter the code in Figure 3. Upon entry, the new input will be loaded into the DSP input circular buffer (in preparation for the MAC function). At this point this New_Input signal could be queued up to be sent over the JTAG interface. After the RTDX API function call was made, the MAC function could be issued (this is essentially the dot product operation), and the result would be stored for a subsequent DSP block. At PC=1010, a call was placed again to output the New_Output using the RTDX_Data_Write.

*To offer some background information on why we emphasize two samples per interrupt is that it is useful to study input-output behavior of these systems. Although in our example an FIR filter was used (whose characteristics can be extensively simulated with packages such as MATLAB), more sophisticated DSP structures like the Least Mean Squared ("LMS") algorithm used in applications such as the echo cancellers in modems and speakerphones*Parker, Steven C. DSP Based Speakerphones for Multimedia System. University at Buffalo M.S. Thesis. February 1997. *are difficult to characterize in simulations as they are so computationally intensive to be easily tested without in-system development.*

It is anticipated that future hard-disk servo designs which rely on higher computational resources, and the algorithms which will require these MIPS will be developed using less science and more art (especially low-cost fixed point solutions). As more advanced algorithms such as LMS, Kalman filtering (observer models), and other adaptive control schemes are used to generate the high performance needed in the next generation of drives the relavance of input-output observation will be of greater importance.

This leads into the inadequacy of RTDX to solve these problems. The current proposed RDTX is too slow to handle full speed DSP emulation. A second problem with the RTDX is in the way the RTDX signal probes are moved. The cycle for using RTDX commands is something like this:

Figure 4: RTDX implementation procedure.

The cycle illustrated in Figure 4 shows that in order to change the signals being monitored, the code must be modified following

1.2.2 Use of OCLA as a DSP Code Development Aid

The OCLA is a completely different approach to the problem of DSP and real-time system observation. Since the OCLA takes the signals internally from within the DSP core - there is *no software intervention.* This is the WYSIWYG of real-time system emulation, unlike the RTDX.

1.3 `Side by side comparison`

RTDX: Offload data through the JTAG port at 8 kb/sec (too slow for at speed real-time operation). TI's C27 DSP offers a visualization port for RTDX with speeds up to 300 kb/s.

OCLA: Offloads data through an innovative 2 wire approach at 1.25 Mbytes/sec. Fast enough for two 256 bit long samples to be downloaded every 10 khz sampling period.

RTDX: Uses DSP software routines to queue up samples to be offloaded to the host system.
OCLA: Uses a completely non-intrusive approach which operates completely outside the operation of the DSP.

1.4 Conclusions

We recognize the fact the RTDX is trying to solve the same problems with real-time system development as the OCLA. While RTDX is an obtrusive software dependent method, the OCLA is a completely non-obtrusive hardware based approach. These are two independent solutions to the problem of real-time system observation.

To answer your question, **"Second you assume it warrants patent protection , why, do we plan to sell this capability to someone ?"**

ABSOLUTELY!! First of all, we have this design included in IBM's CRANE DSP servo controller - where the inclusion of the OCLA dramatically improves bring-up/debug time as well as it enables the code team to develop their real-time software under the exact conditions it will be operating under. Secondly, the design has impressed the Yasu, Japan development team so much it is being incorporated into one of their designs. Finally, we have been approached by Mentor Graphics as to being able to cross-license the OCLA so they can re-sell it to their customers. At least two of their customers have asked about what real-time emulation capability do they have for their DSP cores. Since the original OCLA core was designed for Mentor's clone of TI's C50 DSP, it would be a perfect match for them to be able to resell the OCLA with the DSP.

2. How does the invention solve the problem or achieve an advantage,(a description of "the invention", including figures inline as appropriate)?

3. If the same advantage or problem has been identified by others (inside/outside IBM), how have those others solved it and does your solution differ and why is it better?

4. If the invention is implemented in a product or prototype, include technical details, purpose, disclosure details to others and the date of that implementation.

Fax Transmittal Sheet

From: Steven Parker
Dept 01H / Bld. 321
IBM East Fishkill
1580 Rt. 52
Hopewell Jct. NY 12533

Contact# (914) 892-9288 Tie: 532-9288
Fax # 2-6248

IBM

Microelectronics Division
Development Support Dept.

To: Wonsuck Choi
Co.:

Fax # 703-712-5265
Contact# ___-___-____

Message: 21 pages to follow.

Adhere to Export Procedures for Transmission of Technical data

--> Worsuck Corp
703-712-5265

SYSTEM FOR ANALYSIS OF REAL-TIME HARVARD ARCHITECTURE COMPUTER SYSTEMS

Author: OCLA Team

Contact: **Steven Parker**
IBM Microelectronics Division
1580 Route 52, M/S 5E1
Hopewell Jct., NY 12533
Phone: 914-892-2838 (Tie-Line 532-2838)

ABSTRACT

Two trends in real-time computer systems have caused engineers to re-think the way in which chips are designed. The introduction of digital signal processor with their Harvard style architectures have increased the complexity and difficulty in debugging such as system. Second the introduction of system-on-a-chip architecture with the embedded memories, processors, serial ports, with limited access to the internal busses have also increased the difficulty in developing with such chips.

As such we have developed and implemented a new system to trace internal signals and nodes, as well as to aid the digital signal processing ("DSP") real-time software developer to create and optimize DSP software.

BACKGROUND OF THE INVENTION

Digital signal processing computer systems are becoming increasingly more integrated on a chip which is starting to introduce more problems for debug and emulation schemes. The reasons for these problems comes from a variety of factors, which include increasing of the system speed, VHSIC hardware description language ("VHDL") core based design, and multiple memory blocks.

Real-time microprocessor and digital signal processor based systems pose special problems not present in traditional microprocessor based system. One of the major problems has been in the way these systems are tested and debugged. Traditional emulator schemes such as the JTAG system have relied on the ability to start and stop the processor through the use of a variety of mechanisms. These mechanisms have included setting software breakpoints and single stepping through the machine code. A major draw back of these techniques for real-time systems is the fact the processor must be stopped once a breakpoint is reached or the instruction is executed. In real-time systems it is impractical to stop the processor. For example, in a situation where the DSP is being used as a digital motor controller, stopping the processor could drive an invalid control signal to the actuator with a disastrous result. Therefore test and debug techniques where the processor is not stopped need to be developed.

Hardware debug of real-time digital signal processor systems presents additional problems. The architecture of modern DSPs consists of multiple address and data busses for both program and data memory spaces. Numerous combinations of these busses could be active at any one time. In 'system-on-a-chip' situations where the memory is located on the same chip as the processor core, there is no practical way to view the activity of these busses as you would in a traditional system which has off-chip memory with a logic analyzer. For this reason it is almost impossible to debug a system like this using existing methods.

Development of digital signal processing algorithms also requires special techniques. In many cases the algorithm developer would like to monitor isolated blocks of his or her DSP software design. Reasons for monitoring input-output behavior of these block may be for checking fixed-point performance of the implemented system versus preliminary simulation results, watching for internal overflow or saturation of the MAC within the system, etc. Previous attempts at monitoring these signals have required software hooks to trace these signals, and external processor bus bandwidth, which may or may not be available, to off load them to a PC or workstation. If the signal desired to be monitored changes, it would likely take a code recompile. Therefore it would be good to have a scheme to monitor signals in a DSP without making code changes and without using external bus bandwidth to off load the data.

Taking all of this into account we have developed a new system and associated methods for observing the internal states of a real-time DSP system. This system which we call the on-chip logic analyzer ("OCLA") was the result of trying to address the new problems posed by this new environment.

DISCLOSURE OF THE INVENTION (outline)

Item	Source
Drawings	Taken from the existing bodies of work
Claims	3 Main points below
Prior art - how this affects us	OCLA v. RTDX comparison document
Point by point comparison	OCLA v. RTDX comparison document

ABSTRACT

BACKGROUND OF THE INVENTION

FIGURES

BRIEF DESCRIPTION OF FIGURES

DISCLOSURE OF INVENTION

(1) Introduction / Motivation
    * TBD

(2) Components of the OCLA
    * Description of the diagrams (below)
    * Function descriptions
        Component - Function
        ...
    * GUI Software

(3) Order of operations of the OCLA
    * Initialization, mode selection
    * Trigger mode?
    * Buffer full, time to unload
    * Unloading of the buffer to the host

(4) How it would be used with software / hardware debug
    * DSP system debug
    * System on a chip debug
    * External Sync
    * External Trigger

BRIEF DESCRIPTION OF THE DRAWINGS (outline)

Drawings to include:

-- Main figures --
(1) Overall view of the entire system (VHDL macro, FPGA card, cable, GUI)
(2) Components of the VHDL System (OCLA / Muxes for selection by execution, control, comm, ram)
(3) Example target DSP system
(4) Real-time traces of the main loop (DSP code background)
(5) Multiplexer selection by execution (how / why this works)
(6) Description of the OCLA communication protocol

-- Additional figures --
(7) Command byte structure
(8) Trace word structures
(9) Memory configurations
(10) Overview of the modes of operation

BEST MODE FOR CARRYING OUT THE INVENTION

(should this be a part of the submitted document??)
* yes, not in so many words

1) Real-time trace of DSP software

* Explanation

2) Two-wire communication protocol

* Explanation
* From Joe's original OCLA document - good description, and diagrams
* Needs motivation, why this is a great implementation
  - Limited dedicated I/O available
  - High speed operation for real-time traces

3) Harvard architecture multiplexer selection by execution (data/address busses, io ports)

* Explanation
* From Joe's original OCLA document -
  - How this works, why it works
  - Motivation, current uses - future uses.

4) On-chip logic analyzer (hardware debug)

* Explanation
* Why this is not possible with existing techniques (system on a chip issues)
* What is it for? How is it seen as an overall debug / test strategy

Other items to include:

References
Important figures (number these)
Great deal of detail

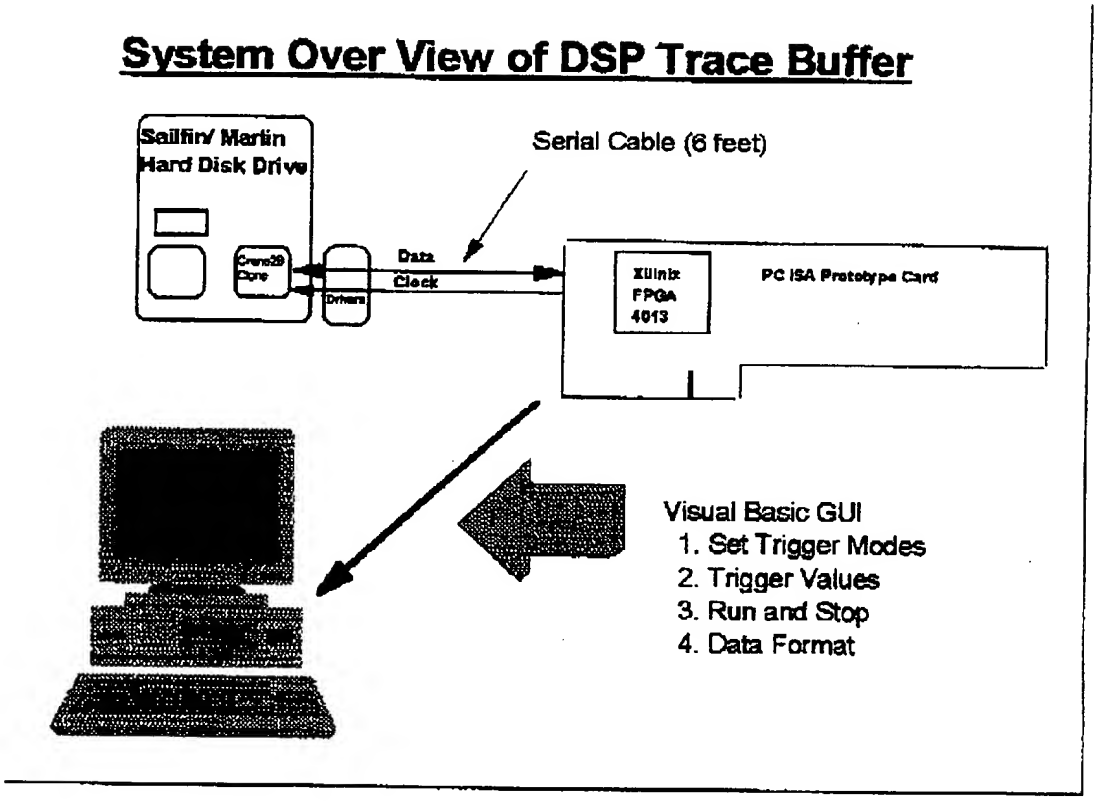


Fig. 1 System Overview

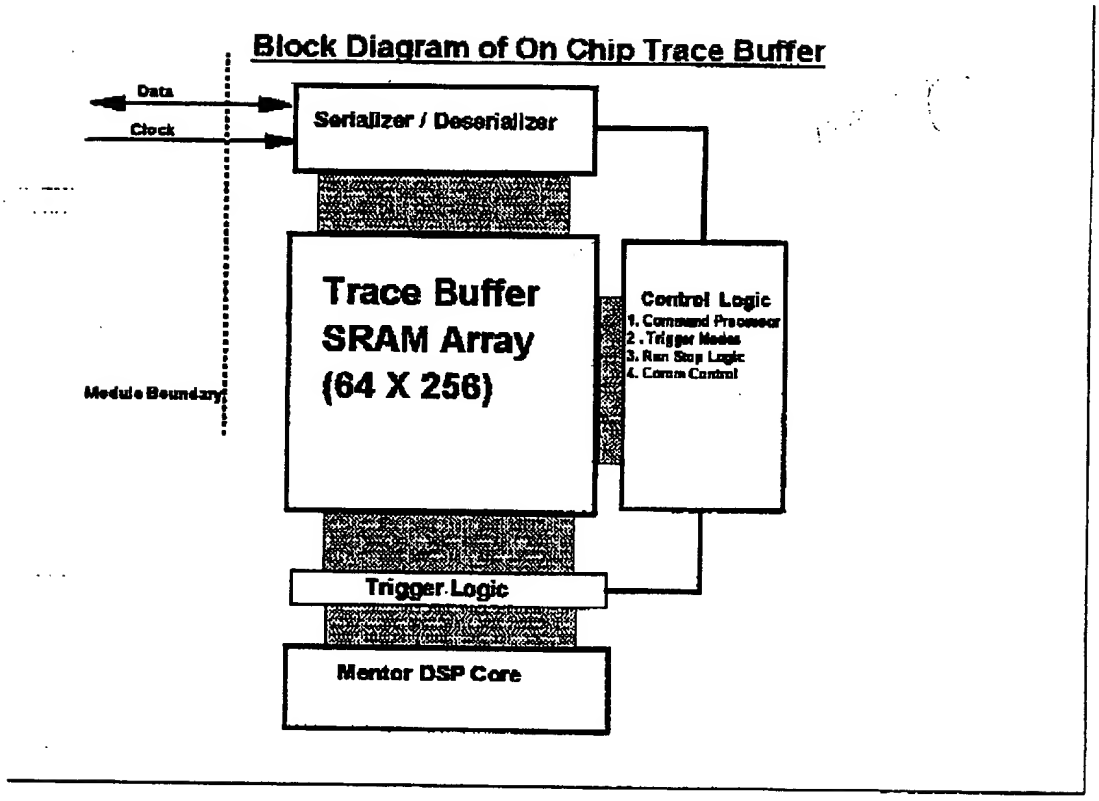
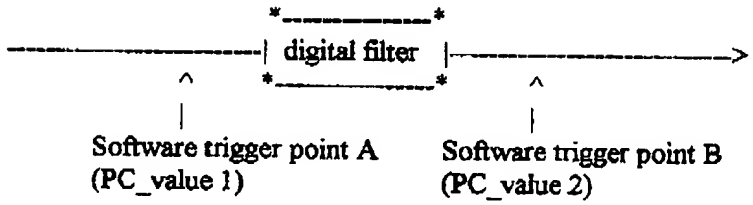


Fig. 2 Block Diagram



Software capture points around a DSP filter structure

To implement this digital filter in code, it might look something like this (FIR filter in pseudo assembly):

PC	Label	Instruction
1000	PC_value_1:	LOAD input into a DSP register (possibly the accumulator)
1002		LOAD starting points of addressing registers
1004		LOAD repeat loop register with N
1006		CLEAR multiply accumulate register
1008		MULTIPLY-ACCUMULATE N times, update pointers
100A		SHIFT to normalize result (if necessary)
100C	PC_value_2:	STORE result

Fig. 3 DSP Signal Flow / Code Example

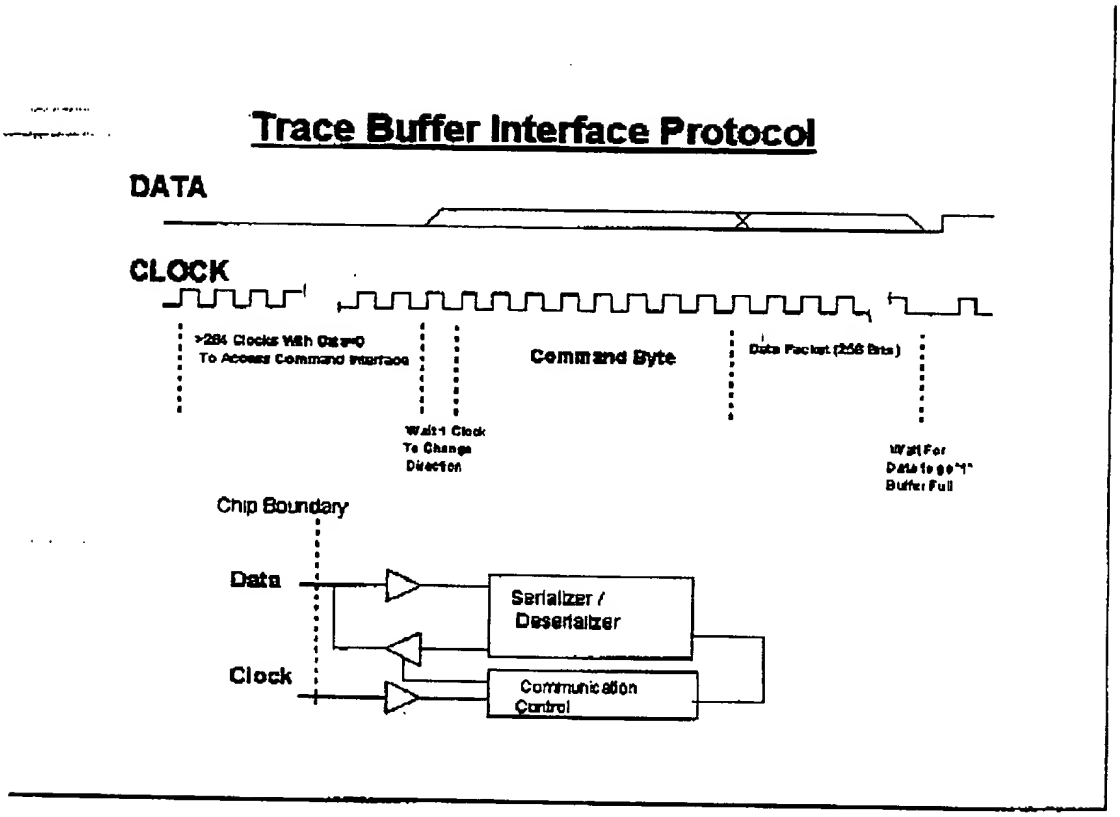


Fig. 4 Communication Protocol

BRIEF DESCRIPTION OF DIAGRAMS

FIG. 1 is a schematic diagram of the high-level overview of the system.
FIG. 2 illustrates the blocks of the custom embedded logic associated with the OCLA.
FIG. 3 illustrates an example target DSP system.
FIG. 4 illustrates the two-wire communication protocol.

DISCLOSURE OF INVENTION (long form)

1.1 High-level overview of the OCLA System

The on-chip logic analyzer ("OCLA") is a system for developing and debugging real-time systems which addresses the problems of observability which were previously unrealizable using previous existing techniques. For these reasons we have developed a new way to bring signals to a host system.

Our implementation of this system is comprised of three main physical components as illustrated in FIG. 1. These components which include the on-chip VHDL macro to be included with the target system, the host system card which includes the interface protocol logic, and the software GUI and API which are used to control the entire system.

The heart of the OCLA system is in the on-chip VHDL macro. This macro is broken down into three main parts as seen in FIG. 2. Control logic orchestrates the operation of the entire VHDL macro, including mode selection and triggering / capturing logic. Communication logic uses an innovative approach to implement a high speed two-wire digital protocol to move data to and from the VHDL macro. The RAM block which is included in this system provides the wide-band memory accesses to store 256 bit / processor cycle.

1.2 Major points / Claims

Portions of the design which we identify to this design as unique, non-obvious inventions are:

* Non-obtrusive, real-time data acquisition for DSP systems
* Two wire protocol for bi-directional communication between host and target systems.
* A multiplexing system for Harvard architecture systems.

2.1 Component Description - VHDL Macro components #1 (Control Logic)

The control logic works to coordinate all operations of the OCLA. It consists of the comparators, state machines, as well as the interface logic to the communication and memory portions of the system. Upon power-up, the control logic loads in the three command words which set up the modes as well as the mask and match settings for the two comparators. Once the "run" bit in the command words is set, the OCLA begins monitoring for the trigger conditions. Depending on the mode the OCLA is currently in the trace buffer begins filling up with either two or 64 trace words. After the buffer has been filled up the control logic indicates to the communication logic to begin off-loading the buffer. Further detail into the operation of the control logic will be provided later in this disclosure.

2.2 Component Description - VHDL Macro components #2 (Communication Logic)

The communication logic supports a sophisticated high speed two-wire target to host interface protocol. Current implementations are running at 10 MHz, with greater speeds expected in the future. Operation of the communication logic works to bring in command words into the OCLA to be decoded and have the mode(s) stored by the control logic. Additionally the communication logic works once a valid trigger condition has been met by the internal signals of the DSP to unload either ~~the either~~ the entire trace buffer or some fraction of the trace buffer. What makes the communication logic so unique is the fact that it is implemented with only two wires. Fig. 4 shows the bi-directional communication protocol used to interact with the OCLA.

2.3 Component Description - VHDL Macro components #3 (Harvard Architecture Multiplexers)

The initial target application of the OCLA was to work with a Texas Instruments ("TI") TMS320C50 DSP chip. This chip is a fixed-point digital signal processor which currently runs at 40 Mhz. Program memory in the C50 is divided into sub-blocks to support a paging scheme for memory access. In addition there are three other small dual-access memories which serve as data memories. One of the requirements for the OCLA was to be able to monitor actions on these internal busses because it would be impractical to use traditional techniques with a logic analyzer.

Single instruction multiple data ("SIMD") DSP instructions such as the ones performed by the C50 require 3 separate busses to be active at any one time. The reason for this is that during a repeated single multiply accumulate like instruction three fetches need to be performed simultaneously. Those three fetches are next program counter ("PC") fetch, data fetch, and coefficient fetch. The architecture of the C50 enables the PC to be from any one of the 5 SARAM blocks; the data fetch could be from one of the DARAM blocks, and the coefficient is from one of the SARAM blocks which is not being used for the PC.

Instead of monitoring the busses on all the memories simultaneously (which would create a very long capture word), through the use of the internal chip-enable (CE) signals we can determine which memory is active and select it to be included in the trace word.

2.4 Component Description - VHDL Macro components #4 (Trace Buffer)

The trace buffer consists of a 256 bit wide by 64 levels deep memory used to store various states of the DSP after some pre-specified condition has been met. This memory is implemented as two separate 128 bit wide by 64 deep. Our memories were unusually wide due to the high bandwidth nature of the operations being performed on the OCLA.

2.5 Component Description - FPGA Card

As part of the host system seen in FIG. 1, the PC ISA interface card is used to communicate with the OCLA. The interface logic on this card is loaded in an FPGA, and a two-wire interface cable runs between the card and the chip with the OCLA macro embedded into it. In addition to the FPGA the card holds a 2 K RAM which stores both the command words loaded and the returned trace data from the OCLA. This RAM serves as a buffer in the real-time modes to synchronize the OCLA and graphical user interface ("GUI").

2.6 Component Description - Software GUI / API

From the user's perspective the OCLA is presented through the use of a Visual Basic GUI. This GUI gives the user an easy way to configure the OCLA as well as a clear view to the data captured by the OCLA

A separate API consisting of many low-level interface routines will be generated to enable support for the real-time modes. Speed limitations of the Visual Basic GUI have caused us to seek alternative methods to off load real-time OCLA data to the host system. Currently we are seeking 40 kb / second throughput to enable the "A then B" to run on a target system with a main loop cycle of 10 MHz. It is anticipated that to achieve these speeds, large buffers will be required on the host system to synchronize OCLA download with file storage.

3.1 Initialization / Bring-up procedure

Once the OCLA and DSP is powered up, the first thing to do is to have the host system load the current mode and associated trigger match and mask values. This is performed by loading in three 256 + 8 bit long trigger words. The first 256 bits represents the data which is as wide as the trace buffer, and the additional 8 bits represents a command word. Loading of the first trigger word sets the 'A' match value, and the second trigger word holds the 'A' mask value. These two of these trigger words are used in the following way: important signals are brought out of the DSP and into the OCLA. These signal go through a set of muxes and into the comparators. Each bit of the trace word from the DSP is then logically ANDed with the mask value and then eXclusive ORed with the match value. This combination creates many different scenarios with which the user can operated the OCLA in.

The third word loaded into the OCLA contains the mode information as well as the B match value. No mask value exists for the B trigger, because if we are to use a mode which

requires a B trigger it is assumed all bits of the B trigger will be used. The third command which is loaded in with the third word will issue to the OCLA to begin running.

3. 2 Modes of Operations

Trigger on A:

In this mode when the "A" trigger is met every cycle of the DSP is stored until the trigger off set register / counter is satisfied. When this condition is satisfied it will cause a buffer full condition to occur. The data signal is set to a "1" requesting the buffer to be emptied by the PC controller. The PC interface will continue to clock until it empties all 64 trace buffer addresses (1.65 ms). When the trace buffer is emptied it will remain inactive until a new command word is received if the auto rearm mode is not selected.

Trigger on A or B:

This mode is similar to the trigger on "A" mode. This enhanced mode can have either the "B" trigger OR an "A" trigger condition met to freeze the capture process when the Off set register is satisfied.

Store only A:

In this mode when the "A" trigger is met only that cycle of the DSP is stored. This process continues until a buffer full condition occurs. The data signal is set to a "1" requesting the buffer to be emptied by the PC controller. The PC interface will continue to clock until it empties all 64 trace buffer addresses. When the trace buffer is emptied it will remain inactive until a new command word is received if the auto rearm mode is not selected.

Store only A Then B:

In this mode when the "A" trigger is met, that cycle of the DSP is captured and queued for transmission as well as arming the "B" trigger. The "A" logic value will be transmitted to the controller while waiting for the "B" trigger to occur. When the "B" trigger is met that cycle of the DSP is captured and queued for transmission as soon as possible. When the auto rearm is enabled the "A" trigger will be rearmed and the process will cycle continuously. This feature will allow continuous monitoring of functions with main loop times of less than 50.2 uS s.

Store only B Then A:

In this mode when the "B" trigger is met, that cycle of the DSP is captured and queued for transmission as well as arming the "A" trigger. The "B" logic value will be transmitted to the controller while waiting for the "A" trigger to occur. When the "A" trigger is met that cycle of the DSP is captured and queued for transmission as soon as possible. When the auto rearm is enabled the "B" trigger will be rearmed and the process will cycle continuously. This feature will allow continuous monitoring of functions with main loop times of less than 50.2 uS. Note that the "B" trigger is just a PC value trigger with out any mask capability.

Store only A OR B:

In this mode when the "A" trigger OR "B" trigger is met, that cycle of the DSP is captured and queued for transmission and will be transmitted to the controller as soon as possible. When

the auto rearm is enabled the "A" OR "B" trigger will be rearmed and the process will cycle continuously. This feature will allow continuous monitoring of functions with main loop times of less than 50.2 uS. Note that the "B" trigger is just a PC value trigger with out any mask capability.

External Synchronization:

In this mode when the specified trigger condition is meet the OCLA_Data pin is pulsed (1 DSP Clock). No data is captured. This mode is used to trigger external devices from the internal specified trigger condition.

Stop & Fill Buffer*:

A single clock received on the OCLA_Clock pin before a buffer full signal (Data = "1" in Run mode) will stop the OCLA and fill the buffer until the off set register is satisfied. When this condition is satisfied it causes a buffer full condition to occur. The data signal is set to a "1" requesting the buffer to be emptied by the PC controller. The PC interface will then continue to clock until it empties all 64 trace buffer addresses (1.65 ms). When the trace buffer is emptied it will remain inactive until a new command word is received even if the auto rearm mode is selected. This mode could be triggered by external circuitry from the controller.

*Please note this mode is NOT on the menu. It is here for reference only.

4.0 How this system would be used

This section of the document will explain how the OCLA is intended to be operated in a real-world environment. Two main cases are presented here; one for hardware debug and one for software debug.

4.1 Hardware Debug

The trigger modes were initially developed to serve as hardware debug modes. In this respect it was intended that specific condition could be realized, and when that condition was met the 64 deep buffer would fill up with important nodes of the DSP. Is this respect the OCLA would be operating more like a traditional logic analyzer. To give the user additional flexibility, we have included an offset register which is loaded in the third OCLA command word. Once a run command has been issued with the OCLA in 'trigger on' mode, the trace memory is treated like a circular buffer which is capturing data once per DSP clock cycle. When a trigger condition is met, the offset counter begins decrementing from the value initially loaded. If the offset counter is not zero it continues capturing in the buffer and decrementing the counter. When the offset counter reaches zero the OCLA indicates to the FPGA card that the operation is complete and a download from the OCLA to the FPGA can begin.

4.2 Software Debug

Store modes were primarily developed for software debug and test. In store modes the user sets up the masks like in the trigger mode. Most digital signal processor software development methodologies will use a block diagram as a high level visualization of the system architecture.

These block diagrams are used for high level software simulation of DSP components, as well as the blueprint for the assembly language coding of the signal flow. The observation was made that specific localized points (nodes) within the DSP system could be trapped on by monitoring the program counter value and looking at the various registers and internal busses within the system. To illustrate this point lets propose that we would like to look at the input-output relationship of a finite impulse response filter deeply embedded within the system (Fig. 3).

After this routine had been coded, it would be necessary to test and optimize it using real data. Since absolute PC locations of the code could be determined from the assembly language linkage process it would easy to make relationships between the two labels PC_value_1 and PC_value_2. Due to the straight line nature of this code, we can expect the operations to occur in the order presented every time (there is no jump/branch operations, so we expect PC_value_1 then PC_value_2).

The first operation at (PC_value_1) would be loading the input value to the filter, and it's placement on one of the data memory busses would be known from the operations. At this point the first software trap is performed, and the complete word is stored in the OCLA trace buffer. As processing of this routine continues, the second PC value (PC_value_2) is trapped and when it is reached the second store occurs.

The processing of real-time DSP code is very different from traditional code. In DSP code, a filter structure like the one used in the example will probably be executed once per analog to digital conversion (and the associated interrupt). In servo and audio applications this interrupt will likely occur at a rate of once per thousands of DSP instruction cycles. For this reason we have plenty of time to off-load the input and output until the next input point is reached again.

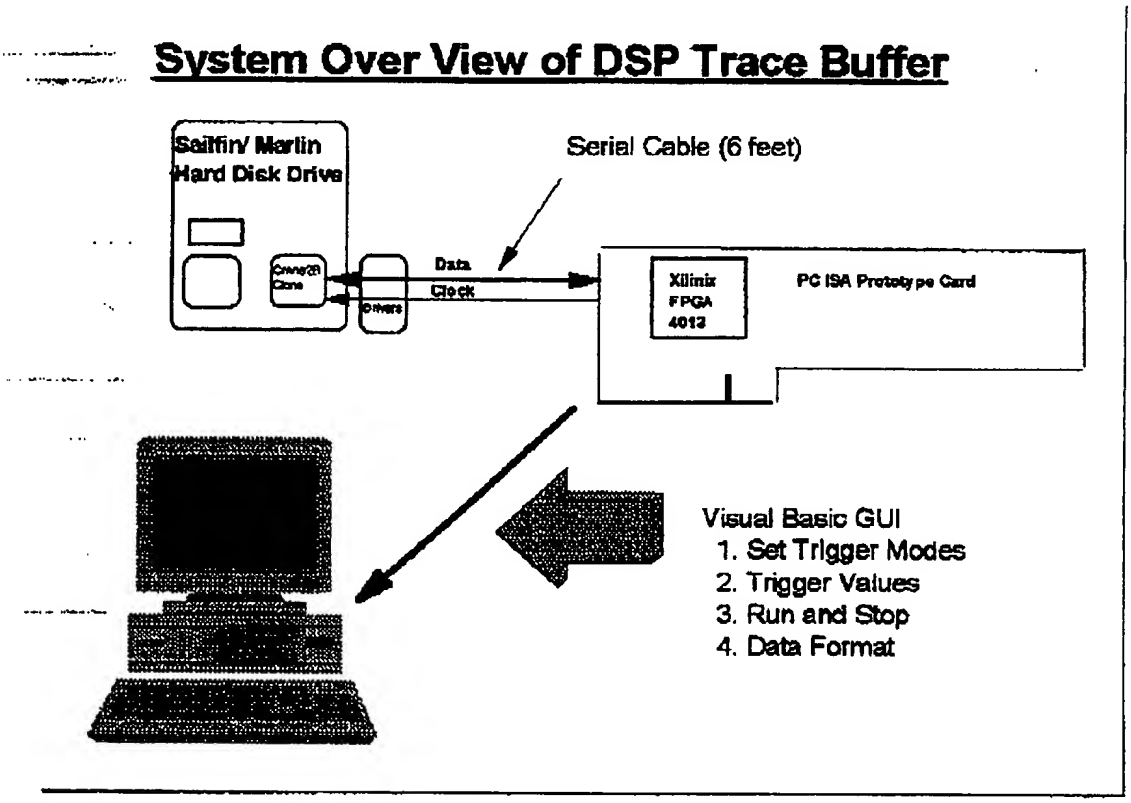
To change the location of these software probes, the developer could simply load the A and B mask values to different program counter values. The advantage of this technique is it does not require the processor to be stopped, and is completely transparent to the execution of the code.

4.3 Additional Modes

In addition to the two major modes listed above, the OCLA also provides the available to allow an external sync signal to be run from the OCLA to an outside hardware analyzer. This signal is set up so that when a trigger condition in met on the chip, a hardware analyzer could be made aware of this condition and perhaps make a capture of board level signals.

One other facility of the OCLA is the use of an external trigger. If the user wanted to use external events to trigger the OCLA, a non-trigger could be loaded in. When a particular condition existed a small clock pulse would be issued to the OCLA causing it to capture the buffer as usual. This mode can also be used as a stop for all the other modes as well.

The OCLA function and data interfaces through the two (2) pins used for the current T.I. emulation modes (EMU0 & EMU1). This 2 pin serial interface will operate at 10 mhz and be controlled by a PC with an adapter card to support this custom interface.



The OCLA interface and control circuits will be reset with the chip power on reset circuit. The OCLA will be ready to receive commands and data at this time. Control to the OCLA can always be established by applying a single OCLA_clock and monitor the OCLA_Data pin for 2 usec. If the OCLA_DATA pin doesn't go to a "1" in that time frame the OCLA was not in the run mode. OCLA_Data can then be driven and a Command/Data word can be sent from the controller. If a "1" is observed on the OCLA_Data pin, the OCLA was in the run mode and was interrupted. The PC interface will then continue to clock until it empties all 256 bits. At this time the data pin will be driven low until the next buffer address is ready then the data pin will be driven high again. The data pin going high will signal the PC to clock out the new 256 bits of data. This process will continue until all 64 trace buffer addresses (1.65 ms) are emptied. At this time the empty trace buffer, will remain inactive and the OCLA_Data pin tristated so a new command word can be received. This occurrs even if the auto rearm mode had been selected.

OCLA requires three 264 bit words to be transmitted from the controller to the OCLA, to be initialized. The interface protocol consists of a command byte followed by 256 bits of data.

The 3 different data streams are:

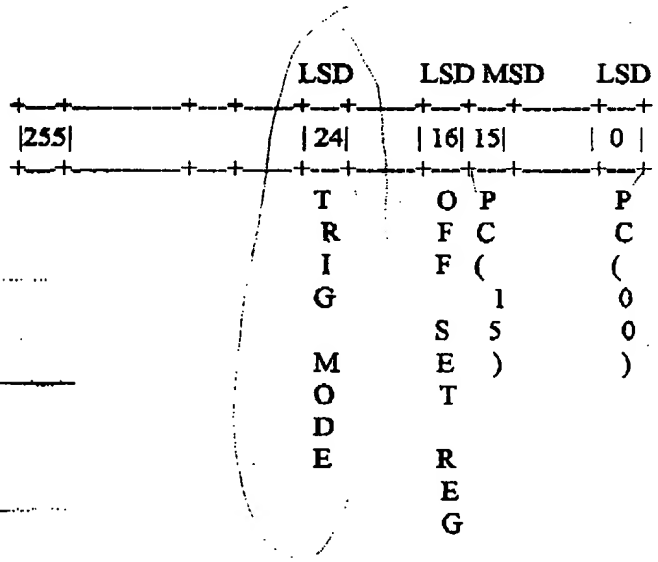
1) Trigger "A" (see previous signal list)

255	249	15	0
NMI		PC(15)	PC(00)

2) Trigger "A" Mask

255	249	15	0
NMI		PC(15)	PC(00)

3) Trigger "B" with Configuration
- -- Program Counter 16 bits
- -- Off Set Reg. (0-63) 2 bits res. 8 bits
- -- Trigger Mode (MODES 1-6) 4 bits



Trigger Modes	4 bits
1) Trigger on A	"0000"h
2) Trigger on B	"0001"h
3) Trigger on A OR B	"0010"h
4) Trigger on A Then B	"0011"h
5) Trigger on B Then A	"0100"h
6) Trigger on A OR B	"0101"h

External Data Bus mux selection	4 bits
Data Write Address & Data Write	"0000"h
External Data Bus mux	"0001"h

The Following Tables define the command byte structure;

COMMAND BYTE STRUCTURE

7	6	5	4	3	2	1	0
RESERVED	READ ADDR RESERVED	DATA DEF (1)	DATA DEF (0)	READ ADDR RESERVED	AUTO REARM	EXT SYNC	RUN

0) Run: This bit when set to a "1" will arm the specified trigger when the command/data word is loaded.

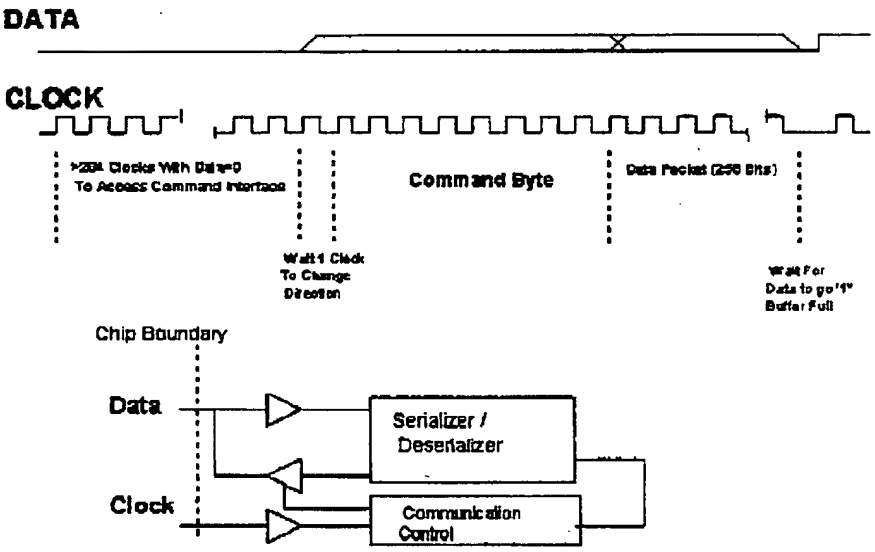
1) Ext Sync: In this mode when the specified trigger condition is meet the OCLA_Data pin is pulsed (1 DSP Clock). No data is captured. This mode is used to trigger external devices from the internal specified trigger condition.

2) Auto Rearm: This bit when set to a "1" will automatically rearm the specified trigger when the buffer has been emptied by the PC controller.

4-5) Data Def: This 2 bits describe the data that will follow the command byte. The possible choices are:

1) Trigger "A" "00"h
2) Trigger "A" Mask "01"h
3) Trigger "B" with Configuration "10"h

Trace Buffer Interface Protocol



Below is an example of the GUI interface for the OCLA. The users manual will be available soon.